Generative models

We want to model the probability distribution of the dataset in order to generate new samples.

Generative models
- Variational AutoEncoders (VAE) : focus on learning latent space structure
- Generative Adversarial Networks (GANs) : focus on learning a distribution no latent space (in general)

AutoEncoders (AE)

The idea behind AE is to produce a representation of the input in a lower-dimensional space, called latent space or latent variables.
The space generated by an autoencoder is good for dimensionality reduction but it is not so good for generation: points in the latent space do not necessarily have similar representations in the original space.

There could be ambiguous points in the latent space that will generate erroneous data.

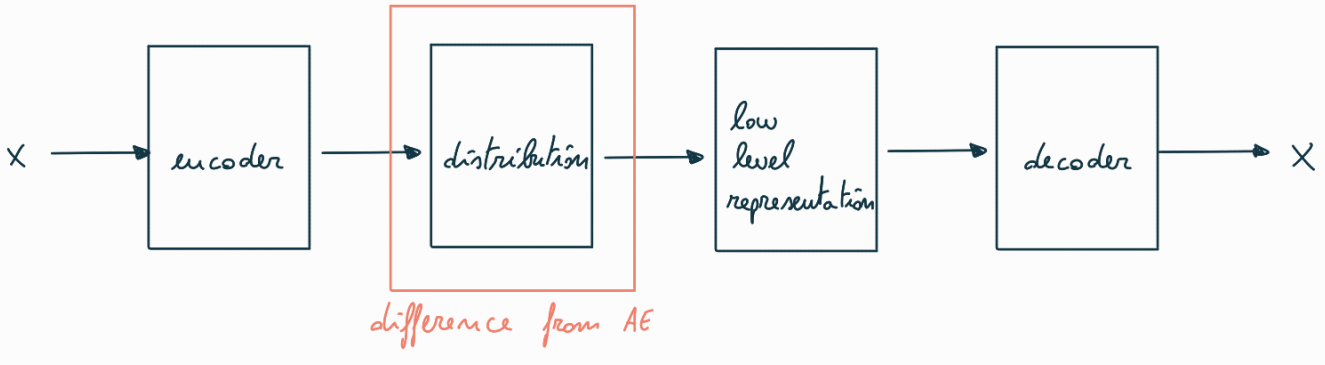
Variational AutoEncoders (VAE)

A Variational AutoEncoder is an AE whose encodings distribution is regularized during training in order to ensure that its latent space has good properties allowing us to generate some new data.

VAE will make sure that the points in the latent space will group into clusters. We will not have just vectors in the latent space, but gaussian distributions. VAE will learn that inputs with similar characteristics must be clustered together.
The latent space of a VAE is represented by a probability distribution

How does VAE work?

We enforce a system to learn a probability distribution rather than a single vector.



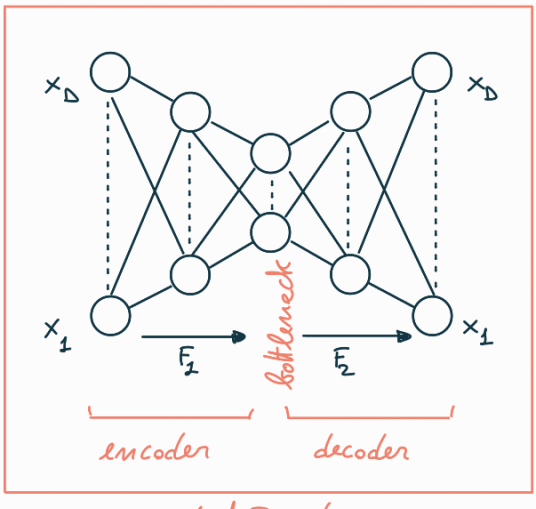
In a standard AE the loss function is given only by the reconstruction error (error = difference between the reconstructed image and the image itself). We usually use the MSE. If the AE is doing a good job (output of the AE is very close to the input, if not the same) the MSE is ≈ 0.
In a variational AE the loss is given by this term (MSE) plus another component, that is the KL difference between two distributions: one is the current distribution that has been computed, $N(z|\mu,\Sigma)$, and the other is the standard distribution, $N(z|0,1)$.

$$KL(N(z|\mu,\Sigma), N(z|0,1))$$

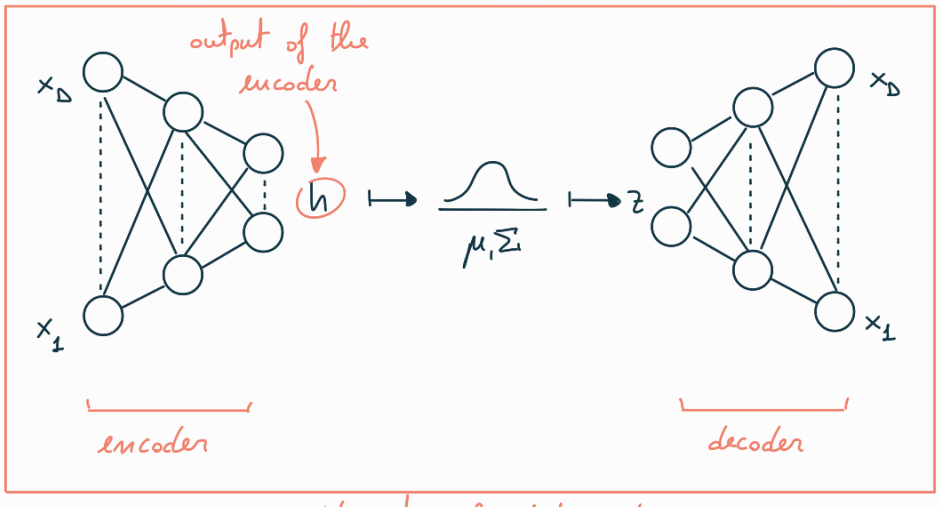
The KL divergence measures how much the latent space differ from a nominal distribution with 0 mean and divergence 1.

The loss for a variational AE is: $MSE + \lambda\, KL(N(z|\mu,\Sigma), N(z|0,1))$
By minimizing these two components we obtain a good tradeoff between a good reconstruction of the output and a good representation of the latent space (missing in the standard AE).

λ weights how much a good representation of the latent space is important. If λ is high you might have a very good representation of the space but loose accuracy in the reconstruction.

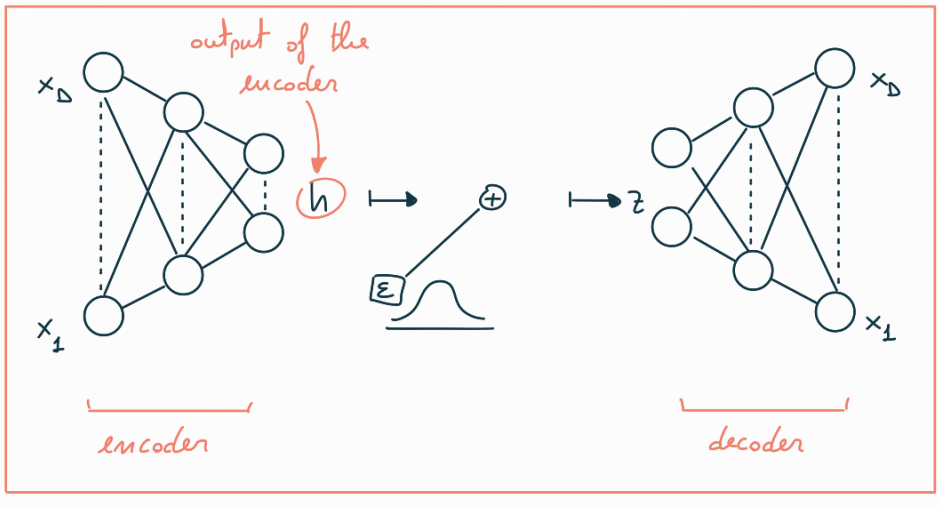


AutoEncoder



Variational AutoEncoder

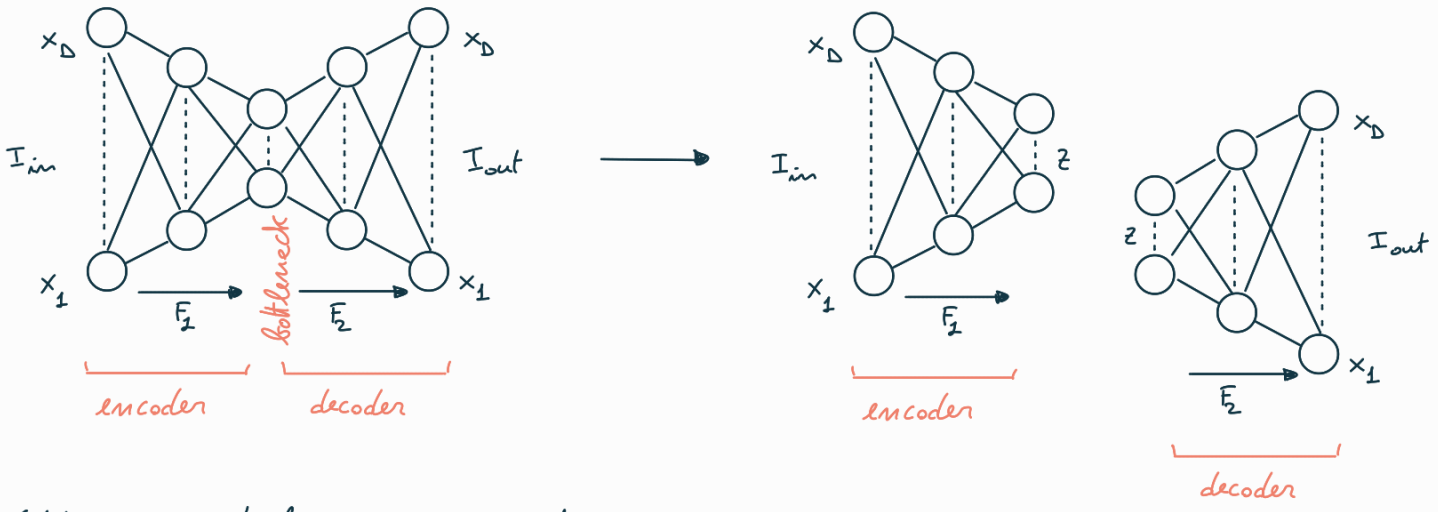
We have problems with backpropagation: the sampling operation is not invertible. When doing backpropagation, given z we don't know how to relate it to h.
This is solved by generating a sample.



When we go forward we have no problem since we can directly apply the gaussian to the output of the encoder
When we have to go backward, we sample from the distribution and once we have a value from the distribution there is no problem with the backpropagation

VAEs are better than AEs in generating meaningful data since we have a better representation of the latent space and we can perturb it.

Generative Adversarial Networks (GANs)



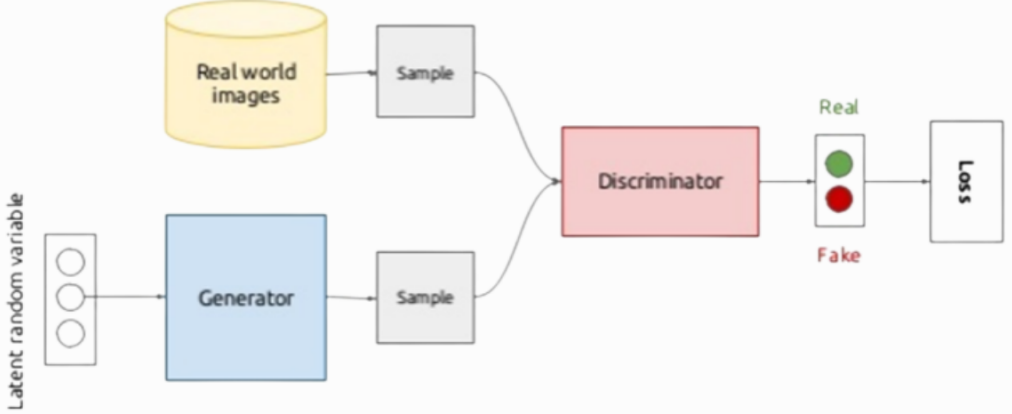
GANs are used for image generation.
The final goal is to generate good data (images) thus we focus on the decoding part.
We can design networks with special features for our tasks.
On top of this, we use Adversarial Training to train the decoder to produce meaningful data.

Adversarial Training ⟶ two pieces of the network are competing with each other to find a good solution. An improvement for one of these components is a disadvantage for the others. In regular networks all the components share the same goal and an improvement for a component is an improvement for all the others.

A GAN is formed by two components

- discriminator
- generator (decoder)

The discriminator is just a binary classifier. It takes as input either an image from the dataset (REAL) or an image produced by the decoder/generator (FAKE) and has to understand if the image is real or not.

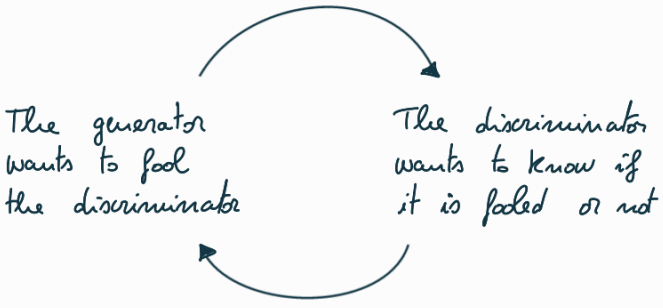


We could start with a generator that is already trained in an autoencoder to speed up the process.

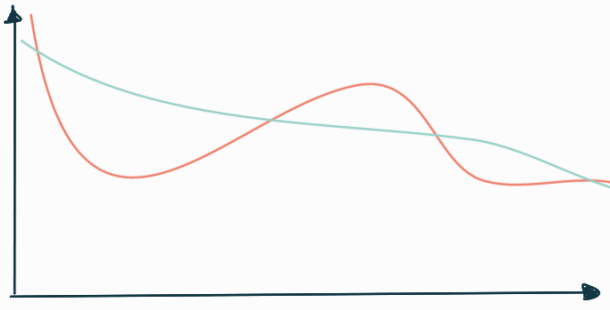
Training procedure:

Repeat:
1. Train the discriminator by keeping the generator fixed
2. Train the generator by keeping the discriminator fixed

When training the discriminator, it knows if the images are REAL or FAKE.
When training the generator, it labels the images that it produces as REAL.
The meaning of the images generated by the generator changes at every step.

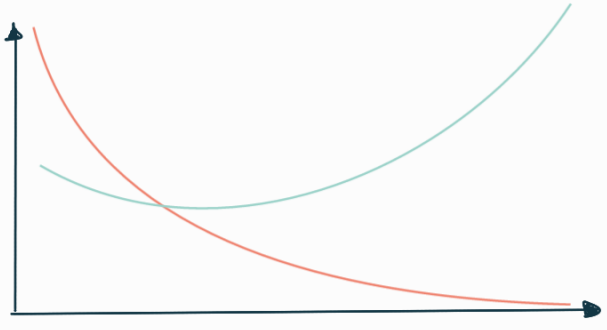


When training the discriminator, the generated images are considered FAKE
When training the generator, the generated images are considered REAL



When the loss for one of the two components increases, it decreases for the other one.

Ideally the two losses will converge.



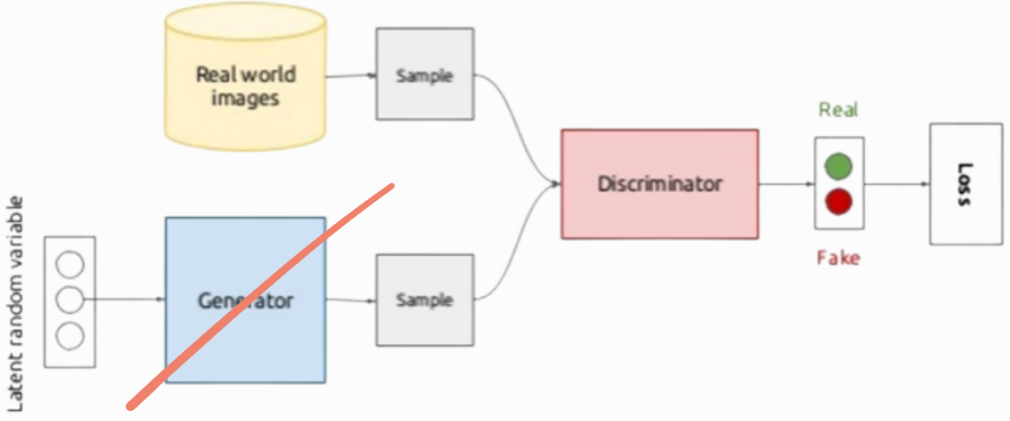
If one loss goes to 0, the other will diverge.
We don't want this to happen

When the two losses are stable (and below some threshold) and the accuracy of the discriminator is around 50% then the model has been trained correctly. It is not easy to train a GAN.
The training usually requires a lot of epochs (the two phases are repeated for each epoch) and it is more difficult than training a CNN or an AE.
As usual we use mini batches (e.g. 128 images) to train the discriminator but here we will divide the batch such that half of that will come from the dataset and the other half will come from the generator;
we will create a standard two-class dataset.

Training procedure:

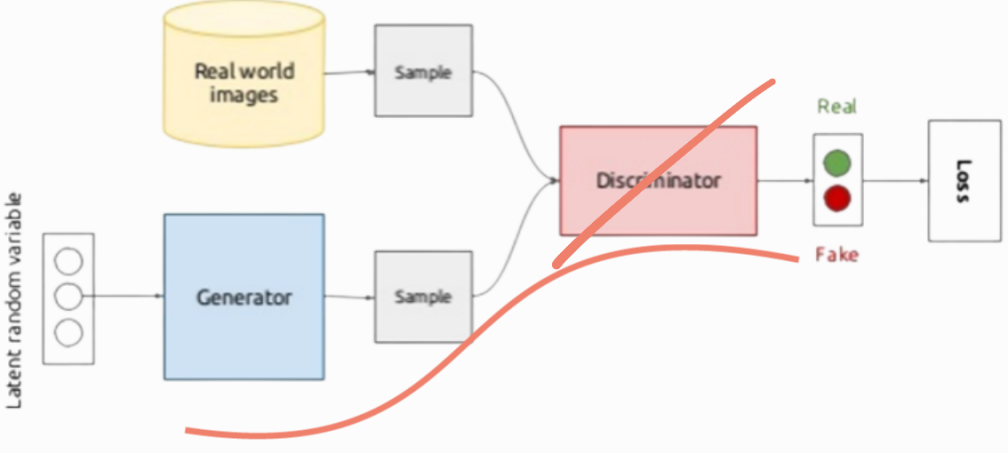
Repeat:

1. Train the discriminator with a batch of data $\{(x_m, \text{REAL})\} \cup \{(x'_m, \text{FAKE})\}$ where x_m comes from the data set, while x_m are images generated from the generator with random values of the latent variable.



The generator is fixed, meaning that the backpropagation is only applied on the discriminator: only the weights of that part of the network are updated

2. Train the generator by using the entire model (generator + discriminator) with discriminator layers fixed (not trainable) with a batch of data $\{(r_k, \text{REAL})\}$, where r_k are random values of the latent variable.



To train the generator we apply the backpropagation on the full model but we do it by keeping the discriminator fixed (freeze its weights).